

FIGURE 1

5 AGTCCCAGACGGGCTTTCCCAGAGAGCTAAAAGAGAAGGGCCAGAGAATGTCGTCCCAG
CCAGCAGGGAACCAAGACCTCCCCGGGCCACAGAGGAACTACTCCTATGGCAGCTGGTAC
ATCGATGAGCCCCAGGGGGCGAGGAGCTCAGCCAGAGGGGAAGTGCCCTCTGCCAC
ACCAGCATACCACCCGGCCTGTACCACGCCCTGCCCTGGCCTCGTCAATCCCTGTGCTG
CTGCTCTGGCATGCTGGAGGGGCCAGCTCTGGCCTGACTGTGCGTGGCAGG
CCCAGGCCCTGCCAGCCCTGTGGATTCTGGCTGGGACAGGCCCGGGCAGTGCCTGCT
10 GCTGTTTCATGGCTCTCTGAGCTCCCTGTTGCTGCTCCCCGACGAGGACGCATTG
CCCTTCTGACTCTGCCCTCAGCACCCAGCCAAAAGATGGGAAACTGAGGCTCCAAGAGGG
GCCTGGAAGAGATACTGGGACTGTTCTATTATGCTGCCCTACTACCCCTGCGCTGCC
GCCACGGCTGGCCACACAGCTGCACACCTGCTGCCAGCAGCACCGCTGTCCGGGCCACCTT
GGGGTCCAGGTCTGGCAGAGGGCAGAGTGTCCCAGGTGCCAAGATCTACAAGTACTAC
15 TCCCTGCTGGCTCCCTGCCCTCCTGCTGGCCTCGGATTCTGAGCCTTGGTACCC
GTGCAGCTGGTGAGAAGCTTCAGCCGTAGGACAGGAGCAGGCTCAAGGGGCTGCAGAGC
AGCTACTCTGAGGAATATCTGAGGAACCTCCCTTGAGGAAGAAGCTGGGAAGCAGCTAC
CACACCTCCAAGCATGGCTCCCTGTCTGGCCCTGCTGCTGAGCAGACTGCATCTAC
ACTCCACAGCCAGGATTCCATCTCCCGCTGAAGCTGGTGCTTCAGACTACACTGACAGGG
20 ACGGCCATTACCAGGTGGCCCTGCTGCTGCTGGTGGCGTGGTACCCACTATCCAGAAG
GTGAGGGCAGGGTCACCACGGATGTCTCCTACCTGCTGGCCGGCTTGGAATCGTGCTC
TCCGAGGACAAGCAGGAGGTGGTGGAGCTGGTAAGCACCATCTGTGGCTCTGGAAAGTG
TGCTACATCTCAGCCTGGTCTGTCTGCTTACTCACCTTGCTGGTCTGATGCGCTCA
CTGGTGACACACAGGACCAACCTTCGAGCTCTGCACCGAGGAGCTGCCCTGGACTTGAGT
25 CCCTTGCATCGGAGTCCCCATCCCTCCGCCAACGCATATTCTGTTGGATGAGCTTCAGT
GCCTACCAGACAGCCTTTATCTGCTTGGCTCTGGTGCAGCAGATCATCTTCTCCTG
GGAACCACGGCCCTGGCCTCTGGTGCATGCCGTGCTCCATGGCAGGAACCTCCTG
CTCTTCCGTTCCCTGGAGTCTCTGTGGCCCTCTGGTGAACTTGGCCCTGGCTGTGATC
CTGCAGAACATGGCAGCCCCATTGGCTCTCTGGAGACTCATGATGGACACCCACAGCTG
30 ACCAACGGCGAGTGCTCATGCAGCCACCTTCTCTCTCCCTCAATGTGCTGGT
GGTGCCATGGTGGCCACCTGGCAGTGCCTCTGCCCCCTACAACGCCATCCACCTT
GGCCAGATGGACCTCAGCCTGCTGCCACCGAGAGGCCACTCTCGACACCCGGCTACTAC
ACGTACCGAAACTCTTGAAGATTGAAGTCAGGCCAGTCGATCCAGCCATGACAGCCTTC
TGCTCCCTGCTCTGCAAGGCCAGAGCCTCTACCCAGGACCATGGCAGCCCCCAGGAC
35 AGCCTCAGACCAGGGAGGAAGACGAAGGGATGCAGCTGCTACAGACAAAGGACTCCATG
GCCAAGGGAGCTAGGCCGGGCCAGCCGGCAGGGCTCGCTGGGTCTGGCTACAG
CTGCTGCACAACCCAACCTGCAGCTTCCGCAAGACGCCCTGTTGGTGCCAATGGT
GCCCCAGCCCTGAGGGCAGGGAGGTCAACCCACCTGCCATCTGCTGAGGCAATGTCC
TGCCTTACCATCCTCCCTCCCTCCGGCTCTCTCCAGCATCACACCAGCCATGCCAGCA
40 GCAGGGTCTCCGGATCACTGTGGTTGGTGGAGGTCTGCTGCACTGGGAGCCTCAGGAG
GGCTCTGCTCCACCACTTGGCTATGGAGAGGCCAGCAGGGTCTGGAGAAAAAAACTG
GTGGGGTAGGGCTTGGTCCAGGAGCCAGTGAGCCAGGGCAGCCACATCCAGGCGTCTC
CCTACCCCTGGCTCGCCATCAGCCTTGAGGGCCTCGATGAAGCCTTCTGGAAACCAC
CCAGCCCAGCTCACCTCAGCCTTGGCCTCACGCTGTGGAGCAGCCAAGGCACTTCCT
45 CACCCCCCTCAGGCCACGGACCTCTGGGAGTGCCGGAAAGCTCCGGTCTCTGGC
CTGCAGGGCAGCCAAAGTCATGACTCAGACCAGGTCCCACACTGAGCTGCCACACTCGA
GAGCCAGATATTTGAGTTTATGCCTTGGCTATTATGAAAGAGGTTAGTGTGTT
CCTGCAATAAAACTGTGCTGAGAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAA

FIGURE 2

MSSQPAGNQTSPGATEDYSYGSWYIDEPQGGEELOPGEVPSCHTSIPPGLYHACLASLS
ILVLLLAMLVRRQLWPDCVRGRPGLPSPVDFLAGDRPRAVPAAVFMVLSSLCLLPD
EDALPFLTLASAPSQDGKTEAPRGAWKILGLFYAAALYYPLAACATAGHTAAHLLGSTLS
WAHLGVQVNQRAECPVPKIYKYYSLASLPLLGLGFLSLWYPVQLVRSFSRRTGAGSK
GLQSSYSEEVLRNLLCRKKLGSSYHTSKHGFLSWARVCLRHCITYTPQPGFHLPLKLVLSA
TLTGTAIYQVALLLVGVVPTIQKVRAVTTDVSYLLAGPGIVLSEDKQEVVELVKHHLW
ALEVCYISALVLSCLLTFLVLMRSLVTHRTNLRALHRGAALDLSPHRSRQAIFCW
MSFSAYQTAFICLGLLVQQIIFFLGTTALAFLVLMPPVHLGRNLLFRSLESSWPFWLTLA
10 LAVILQNMAAHWVLETHDGHQLTNRRVLYAATFLLFPLNVLGAMVATWRVLLSALYN
AIHLGQMDSLLPRAATLDPGYYTYRNFLKIEVSQSHPAMTAFCSLLQAQSLLPRTMA
APQDSLRLPGEDEGMQLLQTKDSMAKGARPGASRGRARWGLAYTLLHNPTLQVFRKTALL
GANGAQP

Important features of the protein:

15 Signal peptide:

None

Transmembrane domain:

20 54-69
102-119
148-166
207-222
25 301-320
364-380
431-451
474-489
560-535

30 Motif file:

Motif name: N-glycosylation site.

35 8-12

Motif name: N-myristoylation site.

40 50-56
176-182
241-247
317-323
341-347
525-531
627-633
45 631-637
640-646
661-667

50 Motif name: Prokaryotic membrane lipoprotein lipid attachment site.

55 364-375

Motif name: ATP/GTP-binding site motif A (P-loop).

55 132-140

FIGURE 3A

PRO XXXXXXXXXXXXXXXXX (Length = 15 amino acids)
Comparison Protein XXXXXYYYYYYY (Length = 12 amino acids)

5

% amino acid sequence identity =

(the number of identically matching amino acid residues between the two polypeptide sequences as determined by ALIGN-2) divided by (the total number of amino acid residues of the PRO polypeptide) =

5 divided by 15 = 33.3%

FIGURE 3B

PRO	XXXXXXXXXX	(Length = 10 amino acids)
Comparison Protein	XXXXXYYYYYYYZZYZ	(Length = 15 amino acids)

5

% amino acid sequence identity =

(the number of identically matching amino acid residues between the two polypeptide sequences as determined by ALIGN-2) divided by (the total number of amino acid residues of the PRO polypeptide) =

10

5 divided by 10 = 50%

FIGURE 3C

PRO-DNA nucleotides)	NNNNNNNNNNNNNNN	(Length	=	14
5 Comparison DNA nucleotides)	NNNNNNL L LLLLLLL	(Length	=	16

% nucleic acid sequence identity =

10 (the number of identically matching nucleotides between the two nucleic acid sequences as determined by ALIGN-2) divided by (the total number of nucleotides of the PRO-DNA nucleic acid sequence) =

6 divided by 14 = 42.9%

15

FIGURE 3D

PRO-DNA	NNNNNNNNNNNN	(Length = 12 nucleotides)
Comparison DNA	NNNNLLLVV	(Length = 9 nucleotides)

5

% nucleic acid sequence identity =

(the number of identically matching nucleotides between the two nucleic acid sequences as determined by ALIGN-2) divided by (the total number of nucleotides of the PRO-DNA nucleic acid sequence) =

10 4 divided by 12 = 33.3%

FIGURE 4A

45

50

55

FIGURE 4B

```

/*
 */
#include <stdio.h>
#include <ctype.h>

5   #define MAXJMP      16      /* max jumps in a diag */
#define MAXGAP      24      /* don't continue to penalize gaps larger than this */
#define J MPS      1024     /* max jmps in an path */
10  #define MX       4       /* save if there's at least MX-1 bases since last jmp */

#define DMAT      3       /* value of matching bases */
#define DMIS      0       /* penalty for mismatched bases */
#define DINS0     8       /* penalty for a gap */
15  #define DINS1     1       /* penalty per base */
#define PINS0     8       /* penalty for a gap */
#define PINS1     4       /* penalty per residue */

20  struct jmp {
    short          n[MAXJMP];    /* size of jmp (neg for delay) */
    unsigned short x[MAXJMP];    /* base no. of jmp in seq x */
    };                           /* limits seq to 2^16 -1 */

25  struct diag {
    int            score;        /* score at last jmp */
    long           offset;       /* offset of prev block */
    short          ijmp;         /* current jmp index */
    struct jmp    jp;           /* list of jmps */
};

30  struct path {
    int            spc;          /* number of leading spaces */
    short          n[J MPS];     /* size of jmp (gap) */
    int            x[J MPS];     /* loc of jmp (last elem before gap) */
};

35  };

40  char          *ofile;        /* output file name */
char          *namex[2];      /* seq names: getseqs() */
char          *prog;          /* prog name for err msgs */
char          *seqx[2];        /* seqs: getseqs() */
int            dmax;          /* best diag: nw() */
int            dmax0;         /* final diag */
int            dna;            /* set if dna: main() */
int            endgaps;        /* set if penalizing end gaps */
45  int            gapx, gapy;   /* total gaps in seqs */
int            len0, len1;     /* seq lens */
int            ngapx, ngapy;   /* total size of gaps */
int            smax;          /* max score: nw() */
int            *xbm;           /* bitmap for matching */
50  long           offset;        /* current offset in jmp file */
struct diag   *dx;            /* holds diagonals */
struct path   pp[2];        /* holds path for seqs */

55  char          *calloc(), *malloc(), *index(), *strcpy();
char          *getseq(), *g_calloc();

```

FIGURE 4C

```
/* Needleman-Wunsch alignment program
 *
 * usage: progs file1 file2
 * where file1 and file2 are two dna or two protein sequences.
 * The sequences can be in upper- or lower-case and may contain ambiguity
 * Any lines beginning with ';' or '<' are ignored
 * Max file length is 65535 (limited by unsigned short x in the jmp struct)
 * A sequence with 1/3 or more of its elements ACGTU is assumed to be DNA
 * Output is in the file "align.out"
 *
 * The program may create a tmp file in /tmp to hold info about traceback.
 * Original version developed under BSD 4.3 on a vax 8650
 */
15 #include "nw.h"
#include "day.h"

static _dbval[26] = {
    1,14,2,13,0,0,4,11,0,0,12,0,3,15,0,0,0,5,6,8,8,7,9,0,10,0
20 };

static _pbval[26] = {
    1, 2|(1<<('D'-'A'))|(1<<('N'-'A')), 4, 8, 16, 32, 64,
    128, 256, 0xFFFFFFFF, 1<<10, 1<<11, 1<<12, 1<<13, 1<<14,
25 1<<15, 1<<16, 1<<17, 1<<18, 1<<19, 1<<20, 1<<21, 1<<22,
    1<<23, 1<<24, 1<<25|(1<<('E'-'A'))|(1<<('Q'-'A'))
};

main(ac, av)
30     int ac;
     char *av[];
{
    prog = av[0];
    if (ac != 3) {
        35     sprintf(stderr, "usage: %s file1 file2\n", prog);
        sprintf(stderr, "where file1 and file2 are two dna or two protein sequences.\n");
        sprintf(stderr, "The sequences can be in upper- or lower-case\n");
        sprintf(stderr, "Any lines beginning with ';' or '<' are ignored\n");
        sprintf(stderr, "Output is in the file \"align.out\"\n");
        exit(1);
    }
    namex[0] = av[1];
    namex[1] = av[2];
    seqx[0] = getseq(namex[0], &len0);
    seqx[1] = getseq(namex[1], &len1);
    xbm = (dna)? _dbval : _pbval;
45

    endgaps = 0;           /* 1 to penalize endgaps */
    ofile = "align.out";   /* output file */

50    nw();                /* fill in the matrix, get the possible jmps */
    readjmps0();            /* get the actual jmps */
    print();                /* print stats, alignment */

55    cleanup(0);          /* unlink any tmp files */
}
```

FIGURE 4D

```

/* do the alignment, return best score: main()
 * dna: values in Fitch and Smith, PNAS, 80, 1382-1386, 1983
 * pro: PAM 250 values
5   * When scores are equal, we prefer mismatches to any gap, prefer
 * a new gap to extending an ongoing gap, and prefer a gap in seqx
 * to a gap in seq y.
 */
nw0
10  {
    char      *px, *py;          /* seqs and ptrs */
    int       *ndely, *dely;     /* keep track of dely */
    int       ndlx, delx;       /* keep track of delx */
    int       *tmp;             /* for swapping row0, row1 */
15   int       mis;              /* score for each type */
    int       ins0, ins1;        /* insertion penalties */
    register  id;               /* diagonal index */
    register  ij;               /* jmp index */
    register  *col0, *coll;      /* score for curr, last row */
    register  xx, yy;           /* index into seqs */

    dx = (struct diag *)g_calloc("to get diag", len0+len1+1, sizeof(struct diag));

    ndely = (int *)g_calloc("to get ndely", len1+1, sizeof(int));
25   dely = (int *)g_calloc("to get dely", len1+1, sizeof(int));
    col0 = (int *)g_calloc("to get col0", len1+1, sizeof(int)),
    col1 = (int *)g_calloc("to get col1", len1+1, sizeof(int)),
    ins0 = (dna)? DINS0 : PINS0;
    ins1 = (dna)? DINS1 : PINS1;

30   smax = -10000;
    if (endgaps) {
        for (col0[0] = dely[0] = -ins0, yy = 1; yy <= len1; yy++) {
            col0[yy] = dely[yy] = col0[yy-1] - ins1;
            ndely[yy] = yy;
        }
        col0[0] = 0;          /* Waterman Bull Math Biol 84 */
    }
    else
40   for (yy = 1; yy <= len1; yy++)
        dely[yy] = -ins0;

    /* fill in match matrix
     */
45   for (px = seqx[0], xx = 1; xx <= len0; px++, xx++) {
        /* initialize first entry in col
         */
        if (endgaps) {
            if (xx == 1)
                col1[0] = delx = -(ins0+ins1);
            else
                col1[0] = delx = col0[0] - ins1;
                ndlx = xx;
        }
50   else {
            col1[0] = 0;
            delx = -ins0;
            ndlx = 0;
        }
    }
60

```

FIGURE 4E

...nw

```
for (py = seqx[1], yy = 1; yy <= len1; py++, yy++) {
    mis = col0[yy-1];
    if (dna)
        mis += (xbm[*px-'A']&xbm[*py-'A'])? DMAT : DMIS;
    else
        mis += _day[*px-'A'][*py-'A'];

    /* update penalty for del in x seq;
     * favor new del over ongoing del
     * ignore MAXGAP if weighting endgaps
     */
    if (endgaps || ndely[yy] < MAXGAP) {
        if (col0[yy] - ins0 >= dely[yy]) {
            dely[yy] = col0[yy] - (ins0 + ins1);
            ndely[yy] = 1;
        } else {
            dely[yy] -= ins1;
            ndely[yy]++;
        }
    } else {
        if (col0[yy] - (ins0+ins1) >= dely[yy]) {
            dely[yy] = col0[yy] - (ins0+ins1),
            ndely[yy] = 1;
        } else
            ndely[yy]++;
    }

    /* update penalty for del in y seq;
     * favor new del over ongoing del
     */
    if (endgaps || ndelx < MAXGAP) {
        if (col1[yy-1] - ins0 >= delx) {
            delx = col1[yy-1] - (ins0+ins1);
            ndelx = 1;
        } else {
            delx -= ins1;
            ndelx++;
        }
    } else {
        if (col1[yy-1] - (ins0+ins1) >= delx) {
            delx = col1[yy-1] - (ins0+ins1);
            ndelx = 1;
        } else
            ndelx++;
    }

    /* pick the maximum score; we're favoring
     * mis over any del and delx over dely
     */
}
```

55

60

FIGURE 4F

...bw

```
id = xx - yy + lenl - 1;
if (mis >= delx && mis >= dely[yy])
    col1[yy] = mis;
5
else if (delx >= dely[yy]) {
    col1[yy] = delx;
    ij = dx[id].ijmp;
    if (dx[id].jp.n[0] && (!dma || (ndelx >= MAXJMP
&& xx > dx[id].jp.x[ij]+MX) || mis > dx[id].score+DINS0)) {
        10
        dx[id].ijmp++;
        if (++ij >= MAXJMP) {
            writejmps(id);
            ij = dx[id].ijmp = 0;
            dx[id].offset = offset;
            offset += sizeof(struct jmp) + sizeof(offset);
        }
        15
    }
    dx[id].jp.n[ij] = ndelx;
    dx[id].jp.x[ij] = xx;
    dx[id].score = delx;
}
20
else {
    col1[yy] = dely[yy];
    ij = dx[id].ijmp;
}
25
if (dx[id].jp.n[0] && (!dma || (ndely[yy] >= MAXJMP
&& xx > dx[id].jp.x[ij]+MX) || mis > dx[id].score+DINS0)) {
    30
    dx[id].ijmp++;
    if (++ij >= MAXJMP) {
        writejmps(id);
        ij = dx[id].ijmp = 0;
        dx[id].offset = offset;
        offset += sizeof(struct jmp) + sizeof(offset);
    }
    35
}
dx[id].jp.n[ij] = -ndely[yy];
dx[id].jp.x[ij] = xx;
dx[id].score = dely[yy];
}
40
if (xx == len0 && yy < lenl) {
    /* last col
     */
    if (endgaps)
        45
        col1[yy] -= ins0+ins1*(lenl-yy);
        if (col1[yy] > smax) {
            smax = col1[yy];
            dmax = id;
        }
        50
}
55
if (endgaps && xx < len0)
    col1[yy-1] -= ins0+ins1*(len0-xx);
    if (col1[yy-1] > smax) {
        smax = col1[yy-1];
        dmax = id;
    }
    60
    tmp = col0; col0 = col1; col1 = tmp;
}
(void) free((char *)ndely);
(void) free((char *)dely);
(void) free((char *)col0);(void) free((char *)col1);}
```

FIGURE 4G

```

/*
 *
 * print() -- only routine visible outside this module
 *
 * static:
 * getmat() -- trace back best path, count matches: print()
 * pr_align() -- print alignment of described in array p[]: print()
 * dumpblock() -- dump a block of lines with numbers, stars: pr_align()
 * nums() -- put out a number line: dumpblock()
 * putline() -- put out a line (name, [num], seq, [num]): dumpblock()
 * stars() -- put a line of stars: dumpblock()
 * stripname() -- strip any path and prefix from a seqname
 */
5
10
15
#include "nw.h"

#define SPC      3
#define P_LINE   256      /* maximum output line */
#define P_SPC    3      /* space between name or num and seq */

20
25
extern  day[26][26];
int     olen;           /* set output line length */
FILE   *fx;             /* output file */

30
35
40
45
50
55
60

print()
{
    int     ix, ly, firstgap, lastgap;      /* overlap */

    if ((fx = fopen(ofile, "w")) == 0) {
        fprintf(stderr, "%s: can't write %s\n", prog, ofile);
        cleanup(1);
    }
    fprintf(fx, "< first sequence: %s (length = %d)\n", namex[0], len0);
    fprintf(fx, "< second sequence: %s (length = %d)\n", namex[1], len1);
    olen = 60;
    ix = len0;
    ly = len1;
    firstgap = lastgap = 0;
    if (dmax < len1 - 1) { /* leading gap in x */
        pp[0].spc = firstgap = len1 - dmax - 1;
        ly -= pp[0].spc;
    }
    else if (dmax > len1 - 1) { /* leading gap in y */
        pp[1].spc = firstgap = dmax - (len1 - 1);
        ix -= pp[1].spc;
    }
    if (dmax0 < len0 - 1) { /* trailing gap in x */
        lastgap = len0 - dmax0 - 1;
        ix -= lastgap;
    }
    else if (dmax0 > len0 - 1) { /* trailing gap in y */
        lastgap = dmax0 - (len0 - 1);
        ly -= lastgap;
    }
    getmat(ix, ly, firstgap, lastgap);
    pr_align();
}

```

FIGURE 4H

```

/*
 * trace back the best path, count matches
 */
5 static
getmat(lx, ly, firstgap, lastgap)
    int      lx, ly;          /* "core" (minus endgaps) */
    int      firstgap, lastgap; /* leading/trailing overlap */
{
10    int      nm, i0, i1, siz0, siz1;
    char     outx[32];
    double   pct;
    register int n0, n1;
    register char *p0, *p1;
15
    /* get total matches, score
     */
20    i0 = i1 = siz0 = siz1 = 0;
    p0 = seqx[0] + pp[1].spc;
    p1 = seqx[1] + pp[0].spc;
    n0 = pp[1].spc + 1;
    n1 = pp[0].spc + 1;

25    nm = 0;
    while (*p0 && *p1) {
        if (siz0) {
            p1++;
            n1++;
            siz0--;
30        }
        else if (siz1) {
            p0++;
            n0++;
            siz1--;
35        }
        else {
            if (xbm[*p0-'A']&xbm[*p1-'A'])
                nm++;
40            if (n0++ == pp[0].x[i0])
                siz0 = pp[0].n[i0++];
            if (n1++ == pp[1].x[i1])
                siz1 = pp[1].n[i1++];
            p0++;
            p1++;
45        }
    }
50
    /* pct homology:
     * if penalizing endgaps, base is the shorter seq
     * else, knock off overhangs and take shorter core
     */
55    if (endgaps)
        lx = (len0 < len1)? len0 : len1;
    else
        lx = (lx < ly)? lx : ly;
    pct = 100.*double(nm)/(double)lx;
    fprintf(fx, "\n");
    fprintf(fx, "< %d match%s in an overlap of %d: %.2f percent similarity\n",
60        nm, (nm == 1)? "" : "es", lx, pct);

```

FIGURE 4I

```

    fprintf(fx, "< gaps in first sequence: %d", gapx);           ...getmat
5     if (gapx) {
        (void) sprintf(outx, " (%d %s%s)",                  +
                      ngapx, (DNA)? "base": "residue", (ngapx == 1)? ":" : "s");
        fprintf(fx, "%s", outx);

10    fprintf(fx, ", gaps in second sequence: %d", gapy);
     if (gapy) {
        (void) sprintf(outx, " (%d %s%s)",                  +
                      ngapy, (DNA)? "base": "residue", (ngapy == 1)? ":" : "s");
        fprintf(fx, "%s", outx);
     }
15    if (DNA)
        fprintf(fx,
                 "\n< score: %d (match = %d, mismatch = %d, gap penalty = %d + %d per base)\n",
                 smax, DMAT, DMIS, DINSO, DJNS1);
     else
20        fprintf(fx,
                 "\n< score: %d (Dayhoff PAM 250 matrix, gap penalty = %d + %d per residue)\n",
                 smax, PINS0, PINS1);
     if (endgaps)
25        fprintf(fx,
                 "< endgaps penalized. left endgap: %d %s%s, right endgap: %d %s%s\n",
                 firstgap, (DNA)? "base": "residue", (firstgap == 1)? ":" : "s",
                 lastgap, (DNA)? "base": "residue", (lastgap == 1)? ":" : "s");
     else
30        fprintf(fx, "< endgaps not penalized\n");

35    static nm;          /* matches in core -- for checking */
    static lmax;         /* lengths of stripped file names */
    static ij[2];        /* jmp index for a path */
    static nc[2];        /* number at start of current line */
    static ni[2];        /* current elem number -- for gapping */
    static siz[2];
    static char *ps[2];   /* ptr to current element */
    static char *po[2];   /* ptr to next output char slot */
40    static char out[2][P_LINE]; /* output line */
    static char star[P_LINE]; /* set by stars() */

/*
 * print alignment of described in struct path pp[]
45 */
static
pr_align()
{
50    int nn;            /* char count */
    int more;
    register i;

    for (i = 0, lmax = 0; i < 2; i++)
        nn = stripname(namex[i]);
55    if (nn > lmax)
        lmax = nn;

        nc[i] = 1;
        ni[i] = 1;
        siz[i] = ij[i] = 0;
        ps[i] = seqx[i];
        po[i] = out[i];

```

FIGURE 4J

```

for (nn = nm = 0, more = 1; more; ) {
    for (i = more = 0; i < 2; i++) {
        /*
         * do we have more of this sequence?
         */
        if (!*ps[i])
            continue;
        more++;
        if (pp[i].spc) { /* leading space */
            *po[i]++ = ' ';
            pp[i].spc--;
        }
        else if (siz[i]) { /* in a gap */
            *po[i]++ = '-';
            siz[i]--;
        }
        else { /* we're putting a seq element
                 */
            *po[i] = *ps[i];
            if (islower(*ps[i]))
                *ps[i] = toupper(*ps[i]);
            po[i]++;
            ps[i]++;
            /*
             * are we at next gap for this seq?
             */
            if (ni[i] == pp[i].x[ij[i]])) {
                /*
                 * we need to merge all gaps
                 * at this location
                 */
                siz[i] = pp[i].n[ij[i]]++;
                while (ni[i] == pp[i].x[ij[i]]))
                    siz[i] += pp[i].n[ij[i]]++;
            }
            ni[i]++;
        }
    }
    if (++nn == olen || !more && nm) {
        dumpblock();
        for (i = 0; i < 2; i++)
            po[i] = out[i];
        nn = 0;
    }
}
/*
 * dump a block of lines, including numbers, stars: pr_align()
 */
static
dumpblock()
{
    register i;
    for (i = 0; i < 2; i++)
        *po[i]-- = '\0';
    ...pr_align
}

```

dumpblock

FIGURE 4K

...dumpblock

```

5      (void) putc('\n', fx);
for (i = 0; i < 2; i++) {
    if (*out[i] && (*out[i] != ' ' || *pos[i] != ' '))
        if (i == 0)
            nums(i);
        if (i == 0 && *out[1])
            stars();
    putline(i);
    if (i == 0 && *out[1])
        fprintf(fx, star);
    if (i == 1)
        nums(i);
}
}

20  /*
 * put out a number line: dumpblock()
 */
static
nums(ix)
25  int      ix;      /* index in out[] holding seq line */
{
    char      nline[P_LINE];
    register  i, j;
    register char  *pn, *px, *py;
30
    for (pn = nline, i = 0; i < lmax + P_SPC, i++, pn++)
        *pn = ' ';
    for (j = nc[ix], py = out[ix]; *py; py++, pn++) {
        if (*py == ' ' || *py == '-')
            *pn = '-';
        else {
            if (i%10 == 0 || (i == 1 && nc[ix] != 1)) {
                j = (i < 0)? -i : i;
                for (px = pn; j /= 10, px--)
                    *px = j%10 + '0';
                if (i < 0)
                    *px = '-';
            }
            else
                *pn = ' ';
            i++;
        }
    }
    *pn = '\0';
50
    nc[ix] = i;
    for (pn = nline; *pn; pn++)
        (void) putc(*pn, fx);
    (void) putc('\n', fx);
}
55

/*
 * put out a line (name, [num], seq, [num]): dumpblock()
 */
static
putline(ix)
60  int      ix;
{

```

FIGURE 4L

```
...putline  
5      int          i;  
      register char *px;  
  
     for (px = namex[ix], i = 0; *px && *px != ':'; px++, i++)  
        (void) putc(*px, fx);  
     for (; i < lmax+P_SPC; i++)  
        (void) putc(' ', fx);  
  
    /* these count from 1:  
     * ni[] is current element (from 1)  
     * nc[] is number at start of current line  
15    */  
    for (px = out[ix]; *px; px++)  
        (void) putc(*px&0x7F, fx);  
    (void) putc('\n', fx);  
20    }  
  
    /*  
     * put a line of stars (seqs always in out[0], out[1]): dumpblock()  
     */  
25    static stars()  
    {  
        int          i;  
        register char *p0, *p1, cx, *px;  
30        if (!*out[0] || (*out[0] == ' ' && *(po[0]) == ' ') ||  
             !*out[1] || (*out[1] == ' ' && *(po[1]) == ' '))  
            return;  
35        px = star;  
        for (i = lmax+P_SPC; i; i--)  
            *px++ = ' ';  
  
        for (p0 = out[0], p1 = out[1]; *p0 && *p1; p0++, p1++) {  
            if (isalpha(*p0) && isalpha(*p1)) {  
40                if (xbm[*p0-'A']&xbm[*p1-'A']) {  
                    cx = '*';  
                    nm++;  
                }  
45                else if (!dma && _day[*p0-'A'][*p1-'A'] > 0)  
                    cx = '_';  
                else  
                    cx = ' ';  
                }  
50                else  
                    cx = ' ';  
                *px++ = cx;  
            }  
55            *px++ = '\n';  
            *px = '\0';  
        }  
    }
```

60

FIGURE 4M

```
/*
 * strip path or prefix from pn, return len: pr_align()
 */
5 static
stripname(pn)
    char    *pn;      /* file name (may be path) */
{
    register char    *px, *py;
10
    py = 0;
    for (px = pn; *px; px++)
        if (*px == '/')
            py = px + 1;
15
    if (py)
        (void) strcpy(pn, py);
    return(strlen(pn));
20
}
25
30
35
40
45
50
55
60
```

stripname

FIGURE 4N

```

/*
 * cleanup() -- cleanup any tmp file
 * getseq() -- read in seq, set dna, len, maxlen
 * g_calloc() -- calloc() with error checkin
 * readjmps() -- get the good jmps, from tmp file if necessary
 * writejmps() -- write a filled array of jmps to a tmp file: nw()
 */
5   #include "nw.h"
10  #include <sys/file.h>

    char  *jname = "/tmp/homgXXXXXX";           /* tmp file for jmps */
    FILE  *fj;

15  int    cleanup();                         /* cleanup tmp file */

    long  lseek();

20  /*
21   * remove any tmp file if we blow
22   */
23  cleanup(i)
24      int    i;
25  {
26      if (fj)
27          (void) unlink(jname);
28      exit(i);
29  }

30  /*
31   * read, return ptr to seq, set dna, len, maxlen
32   * skip lines starting with ';', '<', or '>'
33   * seq in upper or lower case
34   */
35  getseq(file, len)
36      char  *file;    /* file name */
37      int    *len;    /* seq len */
38  {
39      char        line[1024], *pseq;
40      register char  *px, *py;
41      int         natgc, tlen;
42      FILE       *fp;
43
44      if ((fp = fopen(file, "r")) == 0) {
45          fprintf(stderr, "%s: can't read %s\n", prog, file);
46          exit(1);
47      }
48      tlen = natgc = 0;
49      while (fgets(line, 1024, fp)) {
50          if (*line == ';' || *line == '<' || *line == '>')
51              continue;
52          for (px = line; *px != '\n'; px++)
53              if (isupper(*px) || islower(*px))
54                  tlen++;
55      }
56      if ((pseq = malloc((unsigned)(tlen+6))) == 0) {
57          fprintf(stderr, "%s: malloc() failed to get %d bytes for %s\n", prog, tlen+6, file);
58          exit(1);
59      }
60      pseq[0] = pseq[1] = pseq[2] = pseq[3] = '\0';

```

cleanup

getseq

FIGURE 4O

...getseq

```

5      py = pseq + 4;
*len = tlen;
rewind(fp);

10     while (fgets(line, 1024, fp)) {
        if (*line == ';' || *line == '<' || *line == '>')
            continue;
        for (px = line; *px != '\n'; px++) {
            if (isupper(*px))
                *py++ = *px;
            else if (islower(*px))
                *py++ = toupper(*px);
            if (index("ATGCU", *(py-1)))
                natgc++;
        }
        *py++ = '\0';
        *py = '\0';
        (void) fclose(fp);
        dna = natgc > (tlen/3);
        return(pseq+4);
    }

25     char *
g_calloc(msg, nx, sz)
30     {                                /* program, calling routine */
        char *msg;                      /* number and size of elements */
        int nx, sz;
        char *px, *calloc();
35     if ((px = calloc((unsigned)nx, (unsigned)sz)) == 0) {
            if (*msg) {
                fprintf(stderr, "%s: g_calloc() failed %s (n=%d, sz=%d)\n", prog, msg, nx, sz);
                exit(1);
            }
        }
        return(px);
    }

40     /*

45     * get final jmps from dx[] or tmp file, set pp[], reset dmax: main()
46     */
readjmps()
{
    int fd = -1;
    int siz, i0, i1;
    register i, j, xx;

50     if (fj) {
        (void) fclose(fj);
        if ((fd = open(jname, O_RDONLY, 0)) < 0) {
            fprintf(stderr, "%s: can't open() %s\n", prog, jname);
            cleanup(1);
        }
    }
    for (i = i0 = i1 = 0, dmax0 = dmax, xx = len0; ; i++) {
        while (1) {
            for (j = dx[dmax].jmp; j >= 0 && dx[dmax].jp.x[j] >= xx; j--)
;
```

g_calloc

readjmps

FIGURE 4P

...readjmps

```

if (j < 0 && dx[dmax].offset && fj) {
    (void) lseek(fd, dx[dmax].offset, 0);
    (void) read(fd, (char *)&dx[dmax].jp, sizeof(struct jmp));
    (void) read(fd, (char *)&dx[dmax].offset, sizeof(dx[dmax].offset));
    dx[dmax].jmp = MAXJMP-1;
}
else
    break;
}
if (i >= JMPSS) {
    fprintf(stderr, "%s: too many gaps in alignment\n", prog);
    cleanup(1);
}
if (j >= 0) {
    siz = dx[dmax].jp.n[j];
    xx = dx[dmax].jp.x[j];
    dmax += siz;
    if (siz < 0) { /* gap in second seq */
        pp[1].n[i1] = -siz;
        xx += siz;

        /* jd = xx - yy + lenl - 1
         */
        pp[1].x[i1] = xx - dmax + lenl - 1;
        gapy++;
        ngapy -= siz;
    }
    /* ignore MAXGAP when doing endgaps */
    siz = (-siz < MAXGAP || endgaps)? -siz : MAXGAP;
    i1++;
}
else if (siz > 0) { /* gap in first seq */
    pp[0].n[i0] = siz;
    pp[0].x[i0] = xx;
    gapx++;
    ngapx += siz;
}
/* ignore MAXGAP when doing endgaps */
siz = (siz < MAXGAP || endgaps)? siz : MAXGAP;
i0++;

}
else
    break;
}

/* reverse the order of jmps
 */
for (j = 0, i0--; j < i0; j++, i0--) {
    i = pp[0].n[j]; pp[0].n[j] = pp[0].n[i0]; pp[0].n[i0] = i;
    i = pp[0].x[j]; pp[0].x[j] = pp[0].x[i0]; pp[0].x[i0] = i;
}
for (j = 0, i1--; j < i1; j++, i1--) {
    i = pp[1].n[j]; pp[1].n[j] = pp[1].n[i1]; pp[1].n[i1] = i;
    i = pp[1].x[j]; pp[1].x[j] = pp[1].x[i1]; pp[1].x[i1] = i;
}
if (fd >= 0)
    (void) close(fd);
if (fj) {
    (void) unlink(jname);
    fj = 0;
    offset = 0;}}
```

FIGURE 4Q

```
/*
 * write a filled jmp struct offset of the prev one (if any): nw()
5   */
writejmps(ix)
    int      ix;
{
    char    *mktemp0;
10
    if (!fj) {
        if (mktemp(jname) < 0) {
            fprintf(stderr, "%s: can't mktemp0 %s\n", prog, jname);
            cleanup(1);
15
        }
        if ((fj = fopen(jname, "w")) == 0) {
            fprintf(stderr, "%s: can't write %s\n", prog, jname);
            exit(1);
        }
20
        (void) fwrite((char *)&dx[ix].jp, sizeof(struct jmp), 1, fj);
        (void) fwrite((char *)&dx[ix].offset, sizeof(dx[ix].offset), 1, fj);
    }
25
30
35
40
45
50
55
60
```

FIGURE 5

5

GTCGCTCTCCGAGGACAAGCAGGAGGGNGTGGAGCTGGTGAAGCACCATCTGTGGGCTCTG
GAAGTGTGCTACATCTCAGCCTTGGTCTTGTCTGCTTACTCACCTTCCTGGTCTGATG
CGCTCACTGGTGACACACAGGACCAACCTTCGAGCTCTGCACCGAGGAGCTGCCCTGGAC
TTGAGTCCCTTGCATCGGAGTCCCCATCCCTCCGCCAAGCCATAATTCTGTTGGATGAGC
10 TTCAGTGCCTACCAGACAGCCTTTATCTGCCTTGGCTCCTCGTGAGCAGATCATCTTC
TTCCCTGGAAACCACGGCCCTGGCCTTCTGGTGCATGCCTGTGCTCCATGGCAGGAAC
CTCCTGCTCTTCCGTTCCCTGGAGTCCTCGTGGCCCTTCTGGCTGACTTTGGCCCTGGCT
GTGATCTGCAGAACATGGCAGCCCATTGGTCTTCTGGAGACTCATGATGGACACCCA
CAGCTGACCAACCGCGAGTGCTCTATGCAGCCACCTTCTTCTCTTCCCTCAATGTG
15 CTGGTGGGTGCCATGGTGGCACCTGGCGAGTGCTCTCTGCCCTACAACGCCATC
CACCTTGGCCAGATGGACCTCAGCCTGCTGCCACCGAGAGGCCCACTCTGACCCGGC
TACTACACGTACCGAA

FIGURE 6

5 CACAACCAGCCACCCCTCTAGGATCCCAGCCCAGCTGGTCTGGCTCAGAGGGAGAAGGC
CCC GT TGGGAGCACCCCTGCTTGCCTGGAGGGACAAGTTCCGGGAGAGATCAATAAAG
10 GAAAGGAAAGAGACAAGGAAGGGAGAGGT CAGGAGAGCGCTTGATTGGAGGAGAAGGGCC
AGAGAATGTCGTCCCAGCCAGCAGGGAACCAAGACCTCCCCGGGGCACAGAGGACTACT
CCTATGGCAGCTGGTACATCGATGAGCCCCAGGGGGCGAGGAGCTCCAGGCCAGAGGGG
AAGTGCCCTCTGCCACACCAGCATACCACCCGGCTGTACCAACGCCCTGCCTGGCCTCGC
15 TGTCAATCCTTGTGCTGCTCCTGGCATGCTGGT GAGGCGCCGCCAGCTCTGGCCTG
ACTGTGTGCGTGGCAGGCCGGCTGCCAGGCCGGCAGTGCCTGCTGCTGTTTCA
TGGTCCTCCTGAGCTCCCTGTGTTGCTGCTCCCCAGCAGGAGCAGCATTGCCCTCTGA
CTCTGCCCTCAGCACCCAGCCAAGATGGAAAAGT GAGGCTCCAAGAGGGGCTGGAAGA
TA CTGGGACTGTTCTATTATGCTGCCCTACTACCCTCTGGCTGCTGTGCCACGGCTG
20 GCCACACAGCTGCACACCTGCTCGGCAGCACGCTGCTGGGCCACCTTGGGTCCAGG
TCTGGCAGAGGGCAGAGTGTCCCAGGTGCCAAGATCTACAAGTACTACTCCCTGCTGG
CCTCCCTGCCCTCCTGCTGGGCCCTGGATTCCCTGAGCCTTGGTACCCCTGTGCAGCTGG
TGAGAAGCTCAGCGTAGGACAGGAGCAGGCTCCAAGGGGCTGCAGAGCAGCTACTCTG
AGGAATATCTGAGGAACCTCCTTGCAGGAAGAAGCTGGGAAGCAGCTACCACACCTCCA
25 AGCATGGCTTCTGTCTGGGCCCGCTGCTGCTTGAGACACTGCATCTACACTCCACAGC
CAGGATTCCATCTCCCGCTGAAGCTGGTCTTCAGCTACACTGACAGGGACGGCCATT
ACCAGGTGGCCCTGCTGCTGGTGGCGTGGTACCCACTATCCAGAAGGTGAGGGCAG
GGGT CACCACGGATGTCCTACCTGCTGGCCGGCTTGGAAATGTCCTCAGGAGACA
AGCAGGAGGTGGTGGAGCTGGTGAAGCACCATCTGTGGCTCTGGAAAGTGTGCTACATCT
30 CAGCCTTGGTCTTGTCTGCTTACTCACCTTCTGGTCTGATGCGCTCACTGGTACAC
ACAGGACCAAACCTCGAGCTCTGCACCGAGGAGCTGCCCTGGACTTGAGTCCCTGCATC
GGAGTCCCCATCCCTCCGCCAAGCCATATTCTGTTGGATGAGCTTCAGTGCCTACCAGA
CAGCCTTATCTGCCTGGGCTCTGGTGCAGCAGATCATCTTCTTCTGGGAACCACGG
CCCTGGCCTTCTGGTCTCATGCCTGTGCTCATGGCAGGAACCTCTGCTCTTCCGTT
35 CCCTGGAGTCCTCGTGGCCCTTCTGGT GACTTTGGCCCTGGCTGTGATCCTGCAGAAC
TGGCAGCCCATTGGGTCTTCTGGAGACTCATGATGGACACCCACAGCTGACCAACCGGC
GAGT GCTCTATGCAGCACCTTCTTCTTCTGGCCATATTGTGCTGGTGGGTGCCATAG
TGGCCACCTGGCAGTGCTCTCTGCCTCATACACGCCATCCACCTTGGCAGATGG
ACCTCAGCCTGCTGCCACCGAGAGGCCACTCTGACACCCGGCTACTACACGTACCGAA
40 ACTTCTTGAAGATTGAAGTCAGCCAGTCGCATCCAGCCATGACAGCCTCTGCTCCCTGC
TCCTGCAAGCGCAGAGCCTCTACCCAGGACCATGGCAGCCCCCAGGACAGCCTCAGAC
CAGGGGAGGAAGACGAAGGGATGCAGCTGCTACAGACAAAGGACTCCATGGCCAAGGGAG
CTAGGGCCGGGGCCAGCCGGCAGGGCTCGTGGGTCTGGCCTACACGCTGCTGCACA
ACCCACCCCTGCAGGTCTTCCGCAAGACGGCCCTGTTGGGTGCCAATGGTGCCAGCCT
45 GAGGGCAGGGAGGTCAACCCACCTGCCATCTGTGCTGAGGCATGTTCTGCCCTACCA
CTCCTCCCTCCCCGGCTCTCTCCAGCATCACACCAGCCATGCAGCCAGCAGGTCTCC
GGATCACTGTTGGGTGGAGGTCTGCTGACTGGAGCCCTCAGGAGGGCTCTGCTCC
ACCCACTTGGCTATGGGAGAGCCAGCAGGGGTTCTGGAGAAAGAAACTGGTGGGTAGGG
CCTTGGTCCAGGAGCCAGTTGAGCCAGGGCAGCCACATCCAGGCCTCCCTACCCCTGGC
50 TCTGCCATCAGCCTTGAAGGGCCTCGATGAAGCCTCTCTGGAACCAACTCCAGCCCAGCT
CCACCTCAGCCTTGGCCTTCACGCTGTGGAAGCAGCCAAGGCACCTCCTCACCCCTCAG
GCCACGGACCTCTGGGAGTGGCCGAAAGCTCCGGCCTTGGCCTGCAGGGCAG
CCCAAGTCATGACTCAGACCAGGTCCCACACTGAGCTGCCACACTCGAGAGGCCAGATAT
TTTTGTAGTTTATGCCCTTGGCTATTATGAAAGAGGTTAGTGTGTTCCCTGCAATAAA
CTTGTTCCTGAGAAAAA

FIGURE 7

MSSQPAGNQTSPGATEDYSYGSWYIDEPGGEELQPEGEVPSCHTSIPPGLYHACLASL
SILVLLLAMLVRRRQLWPDCVRGRPGLPRPRAVPAAVMVLLSSLCLLPDEDALPFL
5 TLASAPSQDGKTEAPRGAWKILGLFYAALYYPLAACATAGHTAAHLLGSTLSWAHLGV
QWWQRAECPQVPKIYKYSLLASLPLLGLGFLSLWYPQLVRSFSRRTGAGSKGLQSS
YSEEYLRLNLLCRKKLGSSYHTSKHGFLSWARVCLRHCITYTPQPGFHLPLKLVLVSATLTG
TAIYQVALLLLGVVPTIQKVRAVGVTTDVSYLLAGFGIVLSEDKQEVVVELVKHHLWALE
VCYISALVLSCLLTFLVLMRSLVTHRTNLRALHRGAAALDLSPHLRSPHPSRQAIFCWMS
10 FSAYQTAFICLGLLVQQIIFFLGTTALAFLVLMPVLHGRNLLLFRSLESSWPFWTLAL
AVILQNMAAHWFLETHDGPQLTNRRVLYAATFLLFPLNVLGAI VATWRVLLSALYN
AIHLGQMDLSLLPPRAATLDPGYTYRNFLKIEVSQSHPAMTAFCSSLQAQSSLRTM
AAPQDSLRLPGEEDEGMQLLQTKDSMAKGARPGRARWGLAYTLLHNPTLQVFRKTA
LLGANGAQP

15 **Important features of the protein:**

Signal peptide:

none

20 **Transmembrane domain:**

54-71

93-111

140-157

25 197-214

291-312

356-371

425-444

464-481

30 505-522

Motif name: N-glycosylation site.

8-12

35 Motif name: N-myristoylation site.

50-56

167-173

40 232-238

308-314

332-338

516-522

618-624

45 622-628

631-637

652-658

Motif name: Prokaryotic membrane lipoprotein lipid attachment
50 site.

355-366

Motif name: ATP/GTP-binding site motif A (P-loop).

55 123-131

Stra6 Variant Clones

FIGURE 8

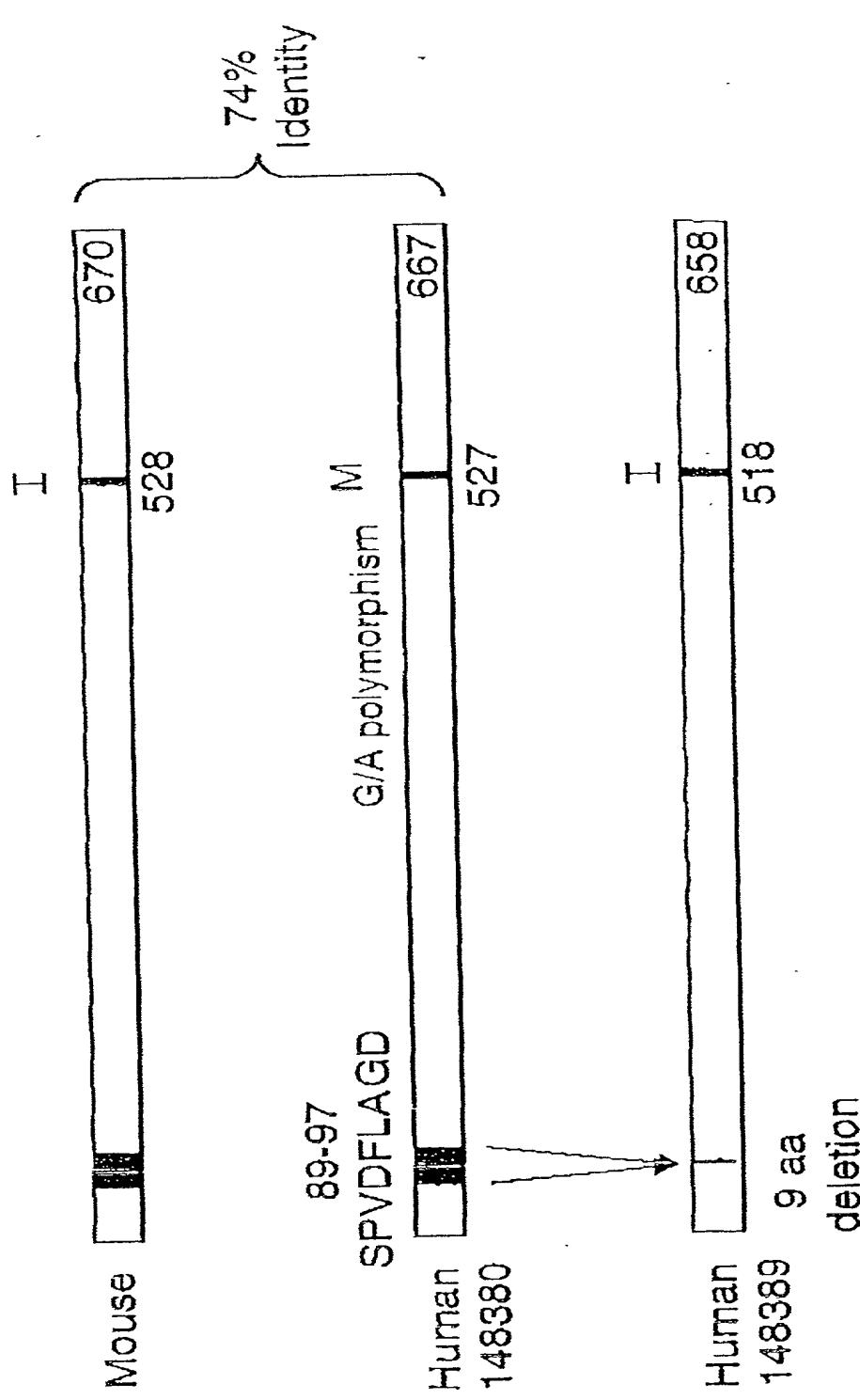
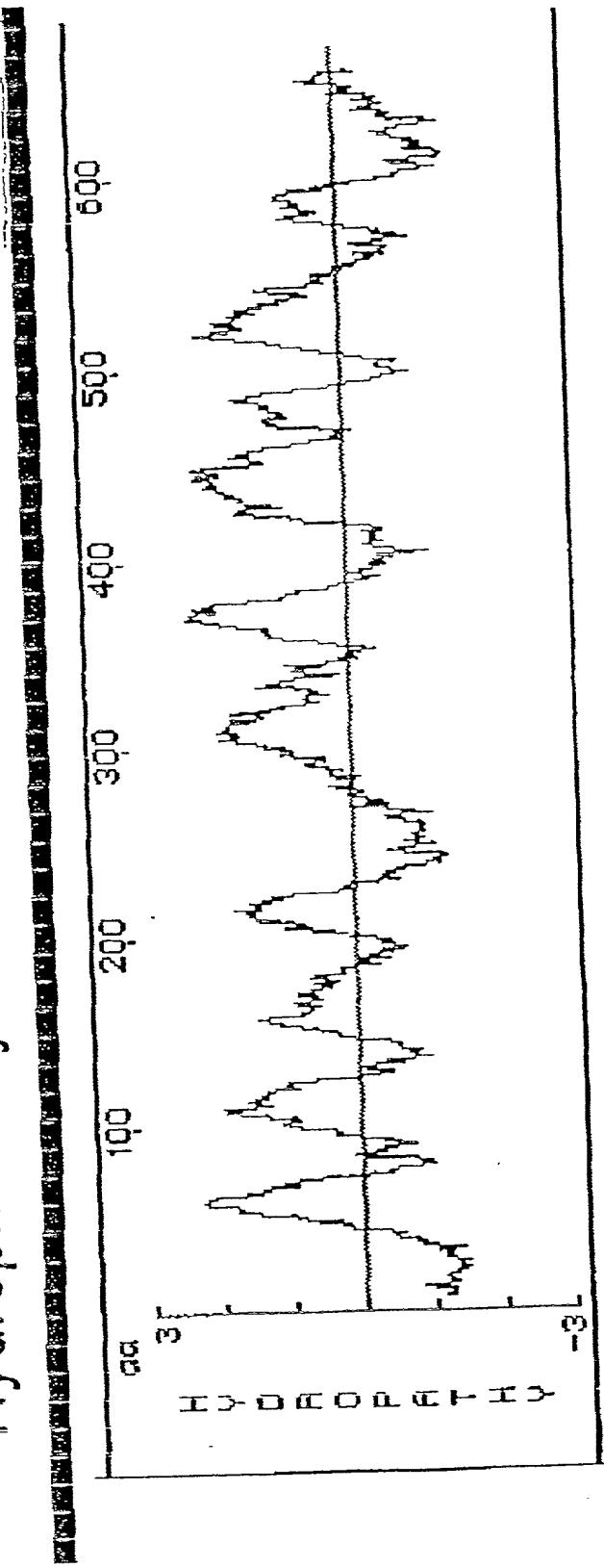


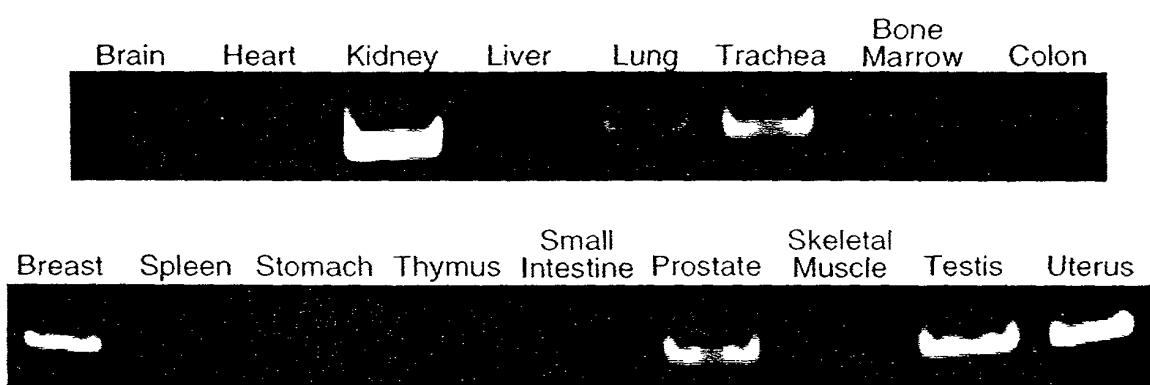
FIGURE 9

Hydropathicity Plot of Human Stra6



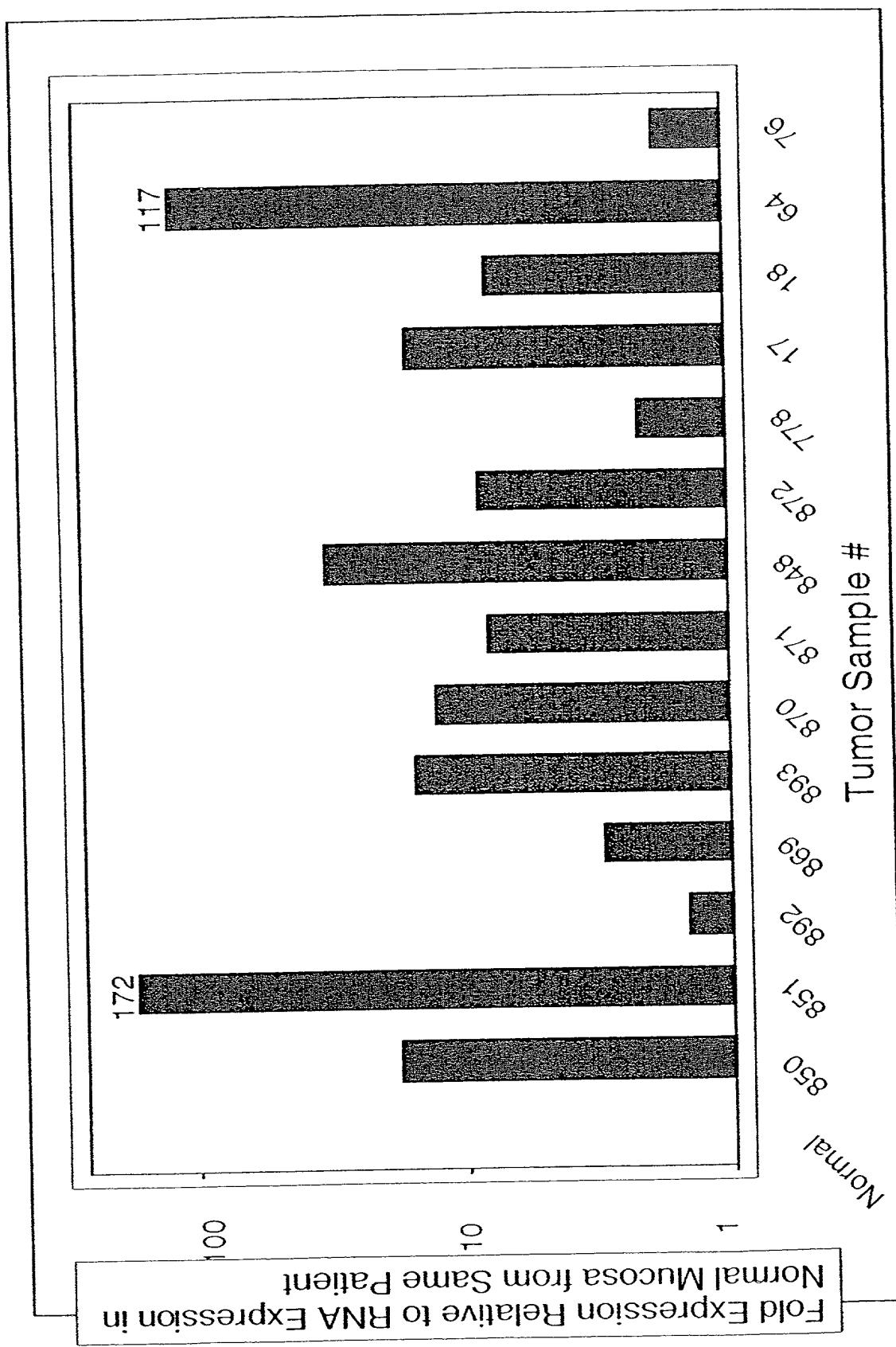
- 3 kb mRNA
- 667 Amino Acids -->50% Residues Hydropophobic
- 73.5 kDa Protein
- 9 Potential Transmembrane Domains

FIGURE 10



Stra6 RNA Expression in Human Colon Tumor Tissue

FIGURE 11



Stra6 RNA Expression in Human Colon Tumor Tissue vs Normal Mucosa From the Same Patient

Taqman Product Analysis After 40 Cycles

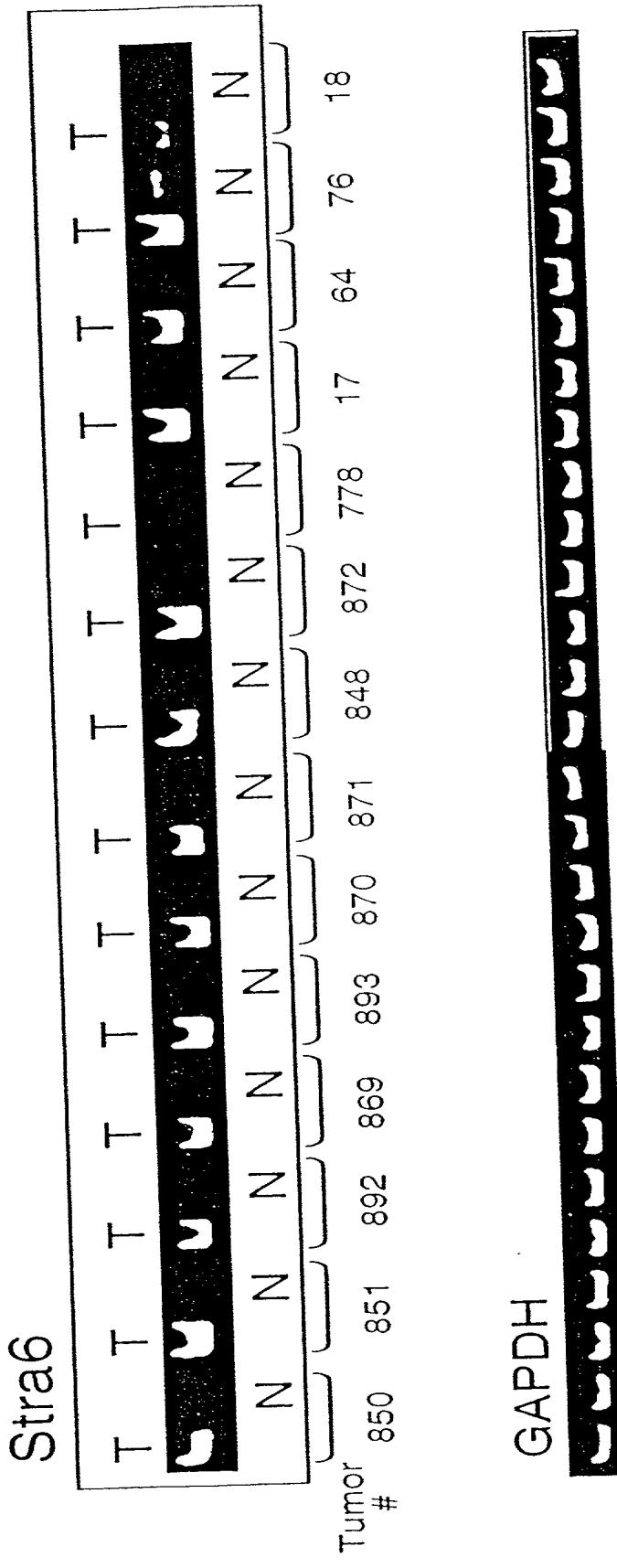


FIGURE 12B

C

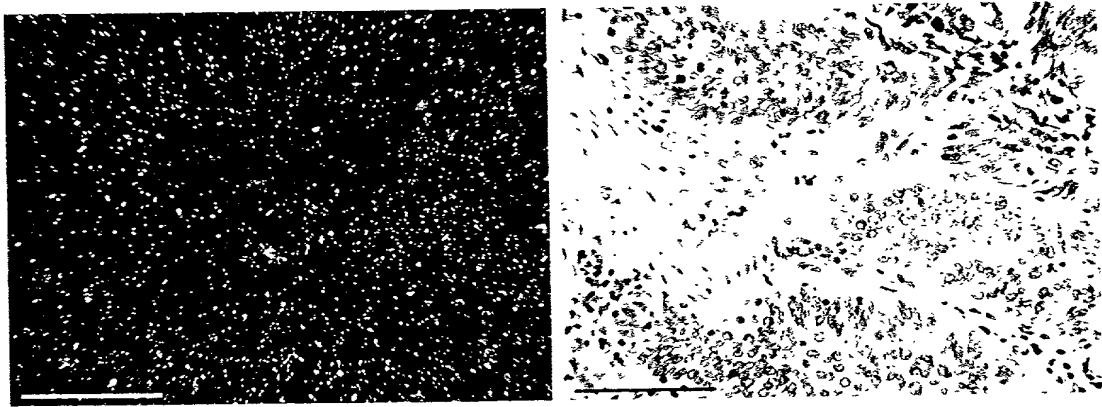
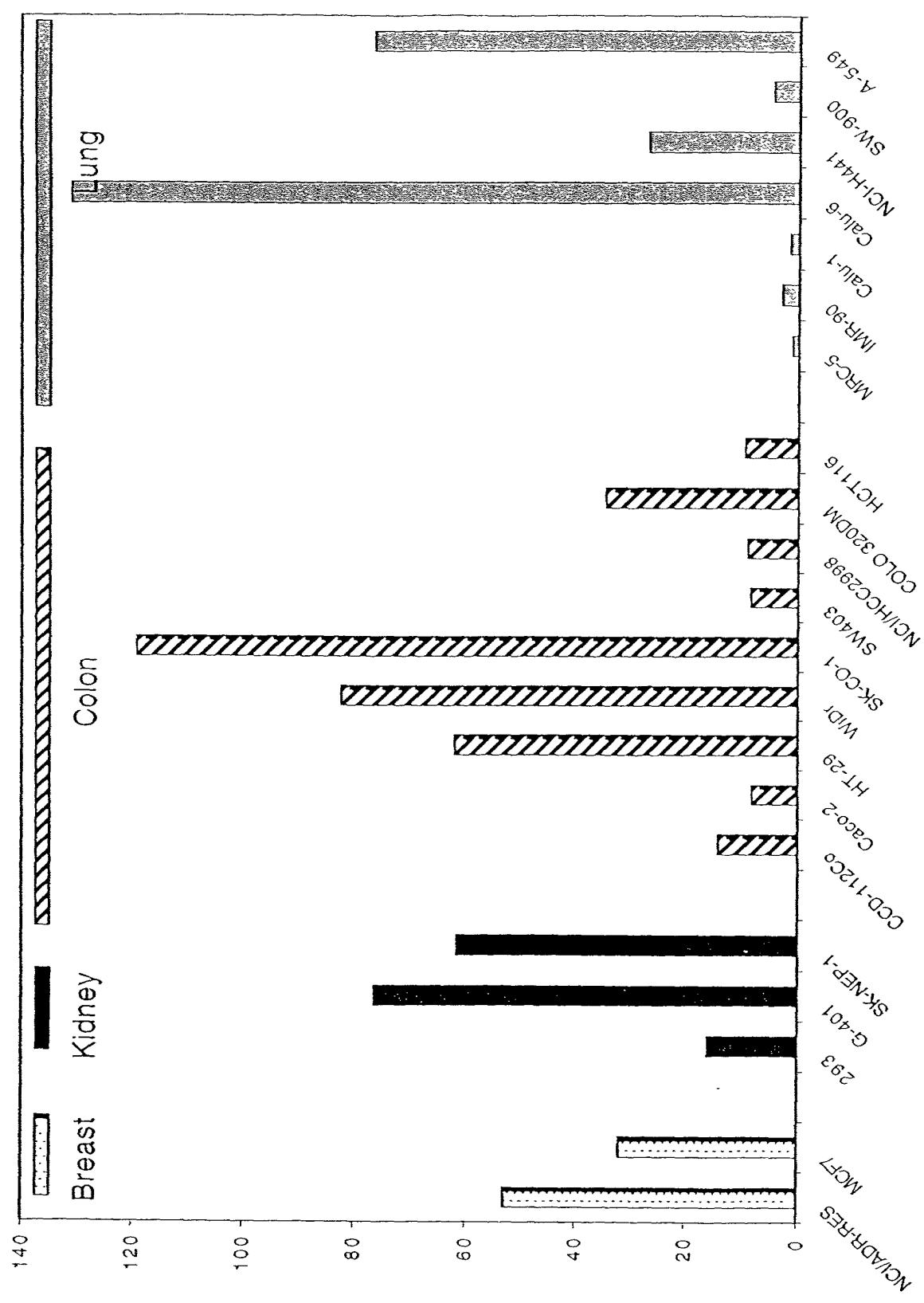


FIGURE 13



Stra6 Peptide Expression in E. coli

Poly-His Cleavable Leader at N-Terminus

FIGURE 14

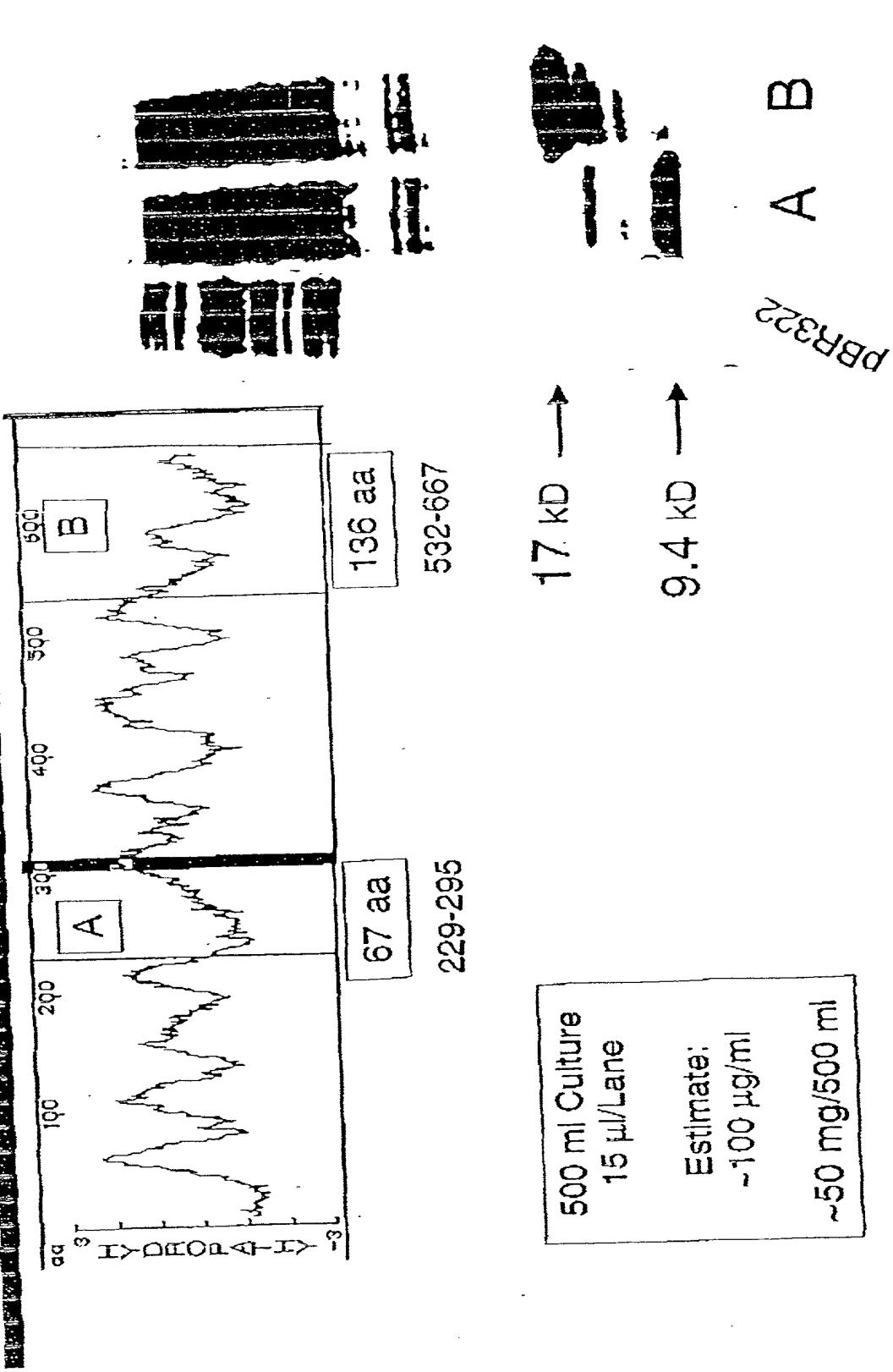


FIGURE 15

Stra6 RNA Expression in Human Colon Carcinoma Cells +/- Retinoic Acid

TM#75 (2/28/00)
VD3 • Vitamin D3 (1 μ M); ATRA • all-trans-retinoic acid (1 μ M)
9cRA • 9-cis-retinoic acid (1 μ M)

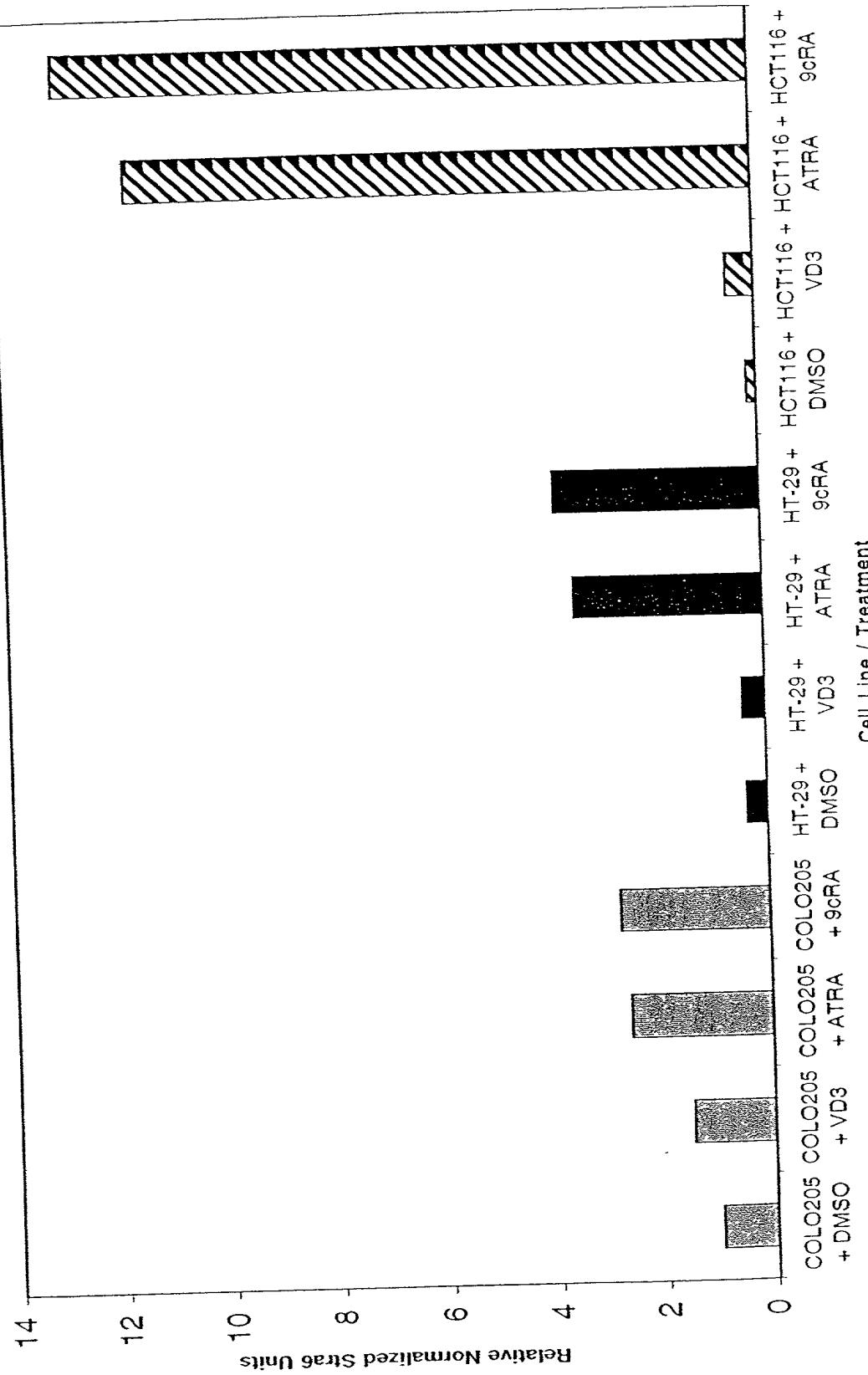


FIGURE 16

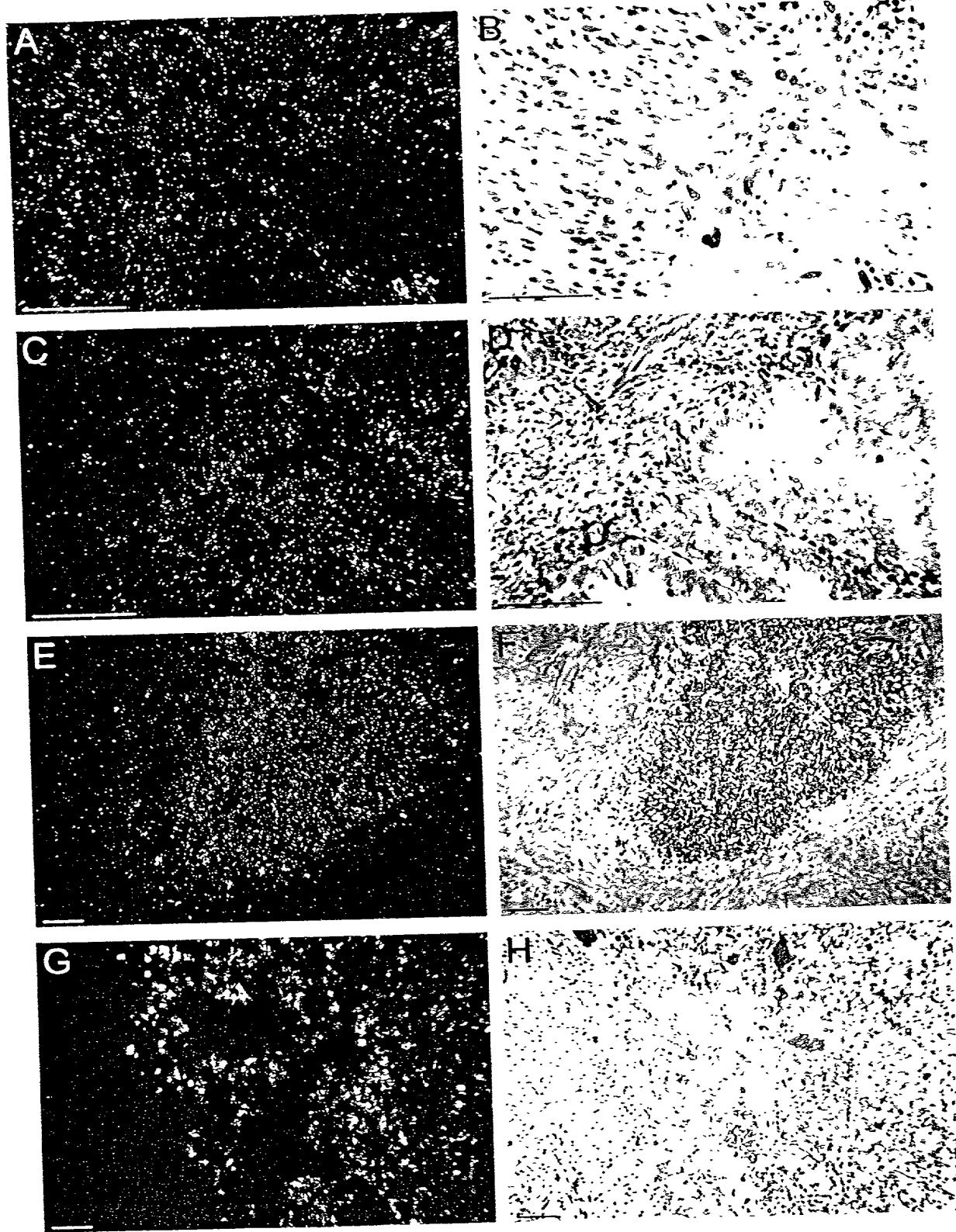


FIGURE 17

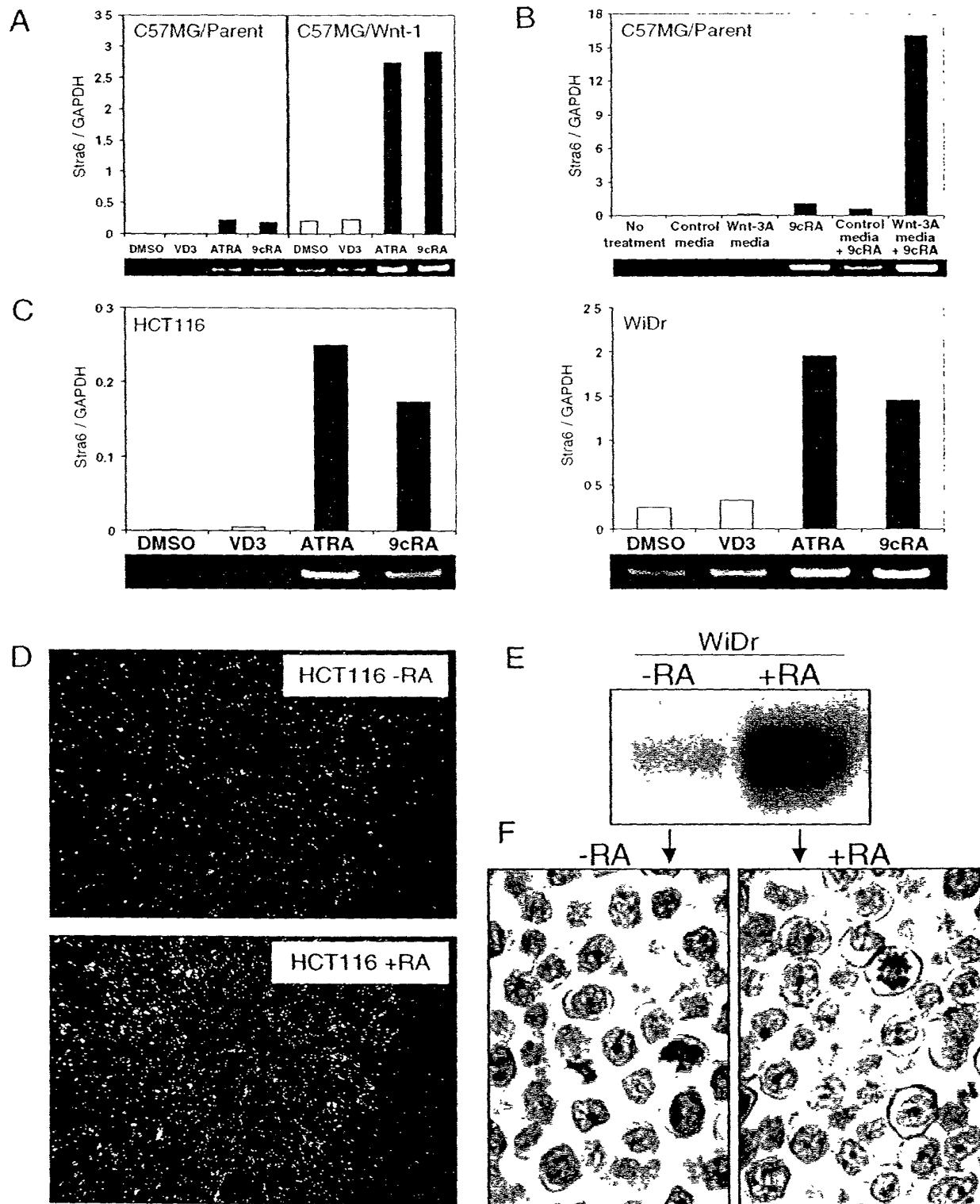
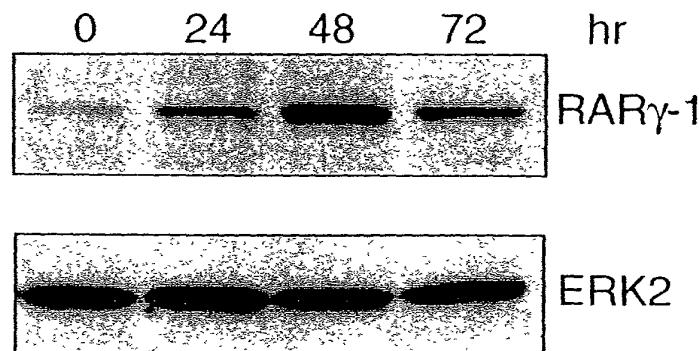


FIGURE 18

A



B

